

Klassenhierarchie §12

Keywords: Vererbung, virtuelle Methoden, Überladen von Methoden, Polymorphismus

Keywords: Basisklasse, abgeleitete Klasse, abstrakte/konkrete Klasse, VMT

Keywords: Basisklassenpointer, `shared_ptr`, `unique_ptr`, dynamic casting

Finales Ziel: STL auf Containern mit polymorphen Elementen. *v_10c_shared/*

Angestelltenhierarchie §12.1

Einführungsbeispiel: Employee, Worker, salesPerson, Manager. *v_9b/* Zur Erinnerung:

- Entwurf (IS-A; HAS-A)
- Initialisierung der Basisklasse via *member initialization list* im Konstruktor der abgeleiteten Klasse.
- Zugriffsrechte: private, protected, public
- Überschreiben von Methoden (*virtual method*).
- Polymorphismus mit Referenz `&Employee` in Parameterliste und VMT zur Auswahl der überschriebenen Methode zur Laufzeit.
Kein Polymorphismus mit Kopie Employee, da erfolgt ein Upcasting (Reduktion auf Eigenschaften der Basisklasse).

Polymorphismus §12.2

- Funktion `printPayment(Employee const&)` in :
 - demonstriert Polymorphismus (VMT).
 - funktioniert nur mit Referenz (oder Pointer); nicht bei Copy (UpCasting).
Copy ist bei abstrakten Basisklassen nicht möglich!
- Example *v_9b_try* mit abgeleiteter Klasse `BoxPromoter`.
 - Dynamic binding §12.2.3
 - Funktionen mit Polymorphismus (`Employee &`) können neue Klasse benutzen, ohne daß diese Funktionen neu übersetzt werden müssen.
 - Vorkompilierte Bibliotheken können einfach genutzt werden.
- Einige Worte zum Casting in Klassenhierarchien:
 - UpCasting per Copy (abgeleitete Klasse → Basisklasse): möglich, da Eigenschaften wegfallen [Compilezeit].

- DownCasting per Copy (Basisklasse → abgeleitete Klasse): nur via spezielle Konstruktoren möglich [Compilezeit].
Fehler im Klassendesign.
- Casting via Referenz: `dynamic_cast<other_class&>(my_instance)`
Fehler über `catch (std::bad_cast& bc)` abfangen [Laufzeit]
- Casting via Pointer: `dynamic_cast<other_class*>(&my_instance)`
Fehler über Test des Ergebnisses auf `nullptr` abfangen [Laufzeit]
- Example `v_9b_cast`
- Allgemein Casting:
 - `static_cast<new_type>(instance)`
 - `const_cast<type>(const type)` [sehr selten benötigt]
 - `dynamic_cast<new_type&>(type&)` in Klassenhierarchien [Spezialfälle]
auch mit Pointer.
 - `reinterpret_cast<new_type>(instance)` auf Ihre Verantwortung [alles zu spät]

STL und Klassenhierarchie §12.3

Polymorphismus funktioniert nur zur **Laufzeit** über **Basisklassenreferenzen** oder **Basisklassenpointer**.

STL-Container mit Referenzen sind nicht möglich, daher müssen wir Basisklassenpointer als Elemente der Container benutzen um den Polymorphismus einer Klassenhierarchie auszunutzen.

- Example `v_9b_poly`:
- Benutze STL-algorithms, aber stets mit Lambda-Funktion (o.ä.) als unären oder binären Operator.
- klassischer Basisklassenpointer `*Employee`
- besser: `shared_ptr<Fahrzeug>` [`v_10c_shared`]
- oder: `unique_ptr<Fahrzeug>` [`v_10c_unique`]

Ein paar interessante Topics

Sichere Pointer

variants_pointer/

- raw pointer; (shallow) copy; explizite Speicherfreigabe, mögliche Mehrfachfreigabe; Copy-Pointer zeigt u.U. auf bereits freigegeben Speicher.
Example: C-array in Funktion (doppelte Freigabe). *dangerous_pointer/*
- `shared_ptr`; (shallow) copy; einfache Freigabe bei letztem Objekt
- `unique_ptr`; nur Referenz möglich; einfache Freigabe

Sichere Pointer

Shallow

- shallow copy einer `struct` (Klasse in der alles public ist); *Ex643-warning.cpp*
Zugriff nach Freigabe des originalen Objektes;
konstante Instanz der `struct` wird durch durch andere Daten überschrieben.
- deep copy einer `struct`; *Ex643-correct.cpp*

Eigene Vektorklasse

myvector/

Eigener raw Pointer auf Daten

- Parameterkonstruktor *myvector.cpp:8*
`new` vs. `new(nothrow)`
- Kopierkonstruktor (deep Copy)
- Zuweisungsoperator (deep Copy): Eigenzuweisung verhindern.
- Demo mit inkorrektem Kopieren: aktiviere `WRONG_CODE` *main.cpp:7*
- Eigener Exception handler für `new` *myexceptions.h:10, main.cpp:26*
`try-throw-catch`
- Index-Check in `operator[]` und `operator[] const`
wirft eigene Exception *myvector.h:98*
definiert in *myexceptions.h:15*
abgefangen in *main.cpp:66*

nochmal `const_cast`

const_cast/

- alte Bibliotheksschnittstelle *processingunit_cpu.hpp*

mutable

Möglichkeit, zur Veränderung (abgeleiteter) Member einer Klasse in einer konstanten Methode

- Flächen-/Umfangsberechnung zur Demo *Class_area_2/*
- Erweiterung um Form Polygon *Class_area_3/*
Polygon_old: no mutable
Polygon: mutable
- Laufzeitvergleich mit großem Container zwischen den beiden Polygonklassen bei Sortierung nach dem Umfang. *Mutable/*

Parallelisierung in der STL

thread_17

Execution policies¹ in C++17

- (extended) Demo Code by Bartłomiej Filipek² *main.cpp*
 - `accumulate` → `reduce`
 - `find`
 - `transform` mit `light_fkt: 1/a`
 - `transform` mit `heavy_fkt: sin(a) * cos(a)`
Memory bandwidth vs. arithmetic limit
- Democode mit `sort` *main_gh.cpp*
- Demos auf Desktop-PC und auf mephisto (> lstopo)

Type Traits and Ranges

thread_17

- Type Traits C++17: *v_8c_cpp17/*
- Type Traits C++20: *v_8c_cpp20/*
- Ranges C++20: *range_demo/*
Container: nimm nur alle geraden Zahlen; und verdopple diese.

¹https://en.cppreference.com/w/cpp/algorithm/execution_policy_tag

²<https://www.cppstories.com/2018/06/parstl-tests/>

Variadic Functions and Templates

shm/zss/

Variable Anzahl von Funktionsparametern oder Templateparametern.
Sehr gute Webseiten von Kuba Sejdak³.

Fun with ASCII⁴:

char_sum.cpp

- How to achieve 103% performance?
- { HARDWORK, KNOWLEDGE, ATTITUDE, BULLSHIT };
- `magic=96` → `magic=0` : BILLGATES (the third!)

Literatur

- [Haase] Gundolf Haase: Einführung in die Programmierung mit C++ (2024), *www*⁵.
- [Stroustrup10] Bjarne Stroustrup: Einführung in die Programmierung mit C++. Pearson Studium, München (2010).
- [Will18] Torsten Will: C++11 Das umfassende Handbuch - Aktuell zu C++17. Rheinwerk Computing (2018)

³<https://kubasejdak.com/variadic-functions-va-args>

⁴<http://www.torsten-horn.de/techdocs/ascii.htm>

⁵http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf